

LETTER

Mutually Exclusive Procedures in Imperative Languages

Keehang KWON[†], *Member*

SUMMARY To represent mutually exclusive procedures, we propose a choice-conjunctive declaration statement of the form *uchoo*(S, R) where S, R are the procedure declaration statements within a module. This statement has the following semantics: request the machine to choose a successful one between S and R . This statement is useful for representing objects with mutually exclusive procedures.

We illustrate our idea via C^{uchoo} , an extension of the core C with a new statement.

key words: *objects, bounded choices, mutual exclusion.*

1. Introduction

Despite the attention, imperative languages [5]–[7] have traditionally lacked mechanisms for representing mutually exclusive tasks. For example, an object like a coffee vending machine is in a superposition state of mutually exclusive procedures, *i.e.*, making a coffee or a tea and require further interactions to determine their final task.

To represent objects with mutually exclusive procedures, we propose to adopt a choice-conjunctive operator in computability logic [1], [2]. To be specific, we allow, within a module or class definition, a choice-conjunctive declaration statement of the form *uchoo*(S_1, \dots, S_n). This statement has the following semantics: request the machine to choose a successful one among S_1, \dots, S_n . This statement is useful for representing mutually exclusive tasks. Examples include function overloading or polymorphic procedures. For example, the *switch* field, declared as

$$uchoo(\text{switch} == \text{on}, \text{switch} == \text{off})$$

indicates that it has two possible values, on and off, and its final value will be determined at run time by the machine.

Another example is the sorting procedure.

$$uchoo(qsort(L) = \dots, heapsort(L) = \dots)$$

This system is in a superposition state of several possible implementations and requires the machine to determine its final implementation.

It can be easily seen that our new statement has many applications in representing most interactive systems. The following declaration represents an interactive object that requires the machine to choose his major and the amount of his tuition.

```
module templeU
uchoo(
  major == english; tuition == $2,000,
  major == medical; tuition == $4,000,
  major == liberal; tuition == $4,000);
```

with the main program

```
read(major);
print(tuition);
```

In the above, the system requests the user to type in a particular major. If the user types in his major, say, medical, then the machine tries to select one among three majors, which leads to a success. After major == medical is selected, the machine sets his tuition to \$4,000 as well. The machine then prints the value of the tuition.

This paper focuses on the minimum core of C. This is to present the idea as concisely as possible. The remainder of this paper is structured as follows. We describe the core Java in Section 2. In Section 3, we present an example of C^{uchoo} . Section 4 concludes the paper.

2. The Language

The language is a subset of the core C with procedure definitions. It is described by G - and D -formulas given by the syntax rules below:

$$G ::= \text{true} \mid A \mid x = v \mid G; G$$

$$D ::= A = G \mid D; D \mid \forall x D \mid uchoo(D_1, \dots, D_n)$$

In the above, A represents a head of an atomic procedure definition of the form $p(t_1, \dots, t_n)$ or a field definition of the form $x == v$ where x is a variable and v is a simple value. A D -formula is called a procedure definition. Note that a boolean condition is a legal statement

Manuscript received January 1, 2003.

Manuscript revised January 1, 2003.

Final manuscript received January 1, 2003.

[†]The authors are with Computer Eng., DongA University. email:khkwon@dau.ac.kr

in our language.

In the transition system to be considered, G -formulas will function as the main statement, and a D -formula enhanced with the machine state (a set of variable-value bindings) will constitute a program. Thus, a program is a union of two disjoint sets, *i.e.*, $\{D\} \cup \theta$ where D is a D -formula and θ represents the machine state. Note that θ is initially set to an empty set and will be updated dynamically during execution via the assignment statements.

We will present an interpreter via a proof theory. Note that this interpreter alternates between the execution phase and the backchaining phase. In the execution phase (denoted by $ex(\mathcal{P}, G, \mathcal{P}')$) it tries to execute a main statement G with respect to a program \mathcal{P} and produce a new program \mathcal{P}' by reducing G to simpler forms until G becomes an assignment statement or a procedure call. The rule (9) and (10) deal with this phase. If G becomes a procedure call or a boolean condition, the interpreter switches to the backchaining mode. This is encoded in the rule (8). In the backchaining mode (denoted by $bc(D, \mathcal{P}, A, \mathcal{P}')$), the interpreter tries to solve a procedure call A and produce a new program \mathcal{P}' by first reducing a procedure definition D in a program \mathcal{P} to simpler forms (via rule (3),(4),(5),(6)) and then backchaining on the resulting definition (via rule (1) and (2)). The notation S seqand R denotes the sequential conjunctive execution of two tasks. To be precise, it denotes the following: execute S and execute R sequentially. It is considered a success if both executions succeed. Similarly, the notation S parand R denotes the parallel conjunctive execution of two tasks. To be precise, it denotes the following: execute S and execute R in parallel. It is considered a success if both executions succeed.

Definition 1. Let G be a main statement and let \mathcal{P} be a program. Then the notion of executing $\langle \mathcal{P}, G \rangle$ successfully and producing a new program \mathcal{P}' – $ex(\mathcal{P}, G, \mathcal{P}')$ – is defined as follows:

- (1) $bc(p(t_1, \dots, t_n) = G_1), \mathcal{P}, p(t_1, \dots, t_n), \mathcal{P}_1)$ if $ex(\mathcal{P}, G_1, \mathcal{P}_1)$. % A matching procedure for $p(t_1, \dots, t_n)$ is found.
- (2) $bc(x == v = G_1), \mathcal{P}, x == v, \mathcal{P}_1)$ if $ex(\mathcal{P}, G_1, \mathcal{P}_1)$. % a boolean conditional statement.
- (3) $bc(\forall x D, \mathcal{P}, A, \mathcal{P}_1)$ if $bc([t/x]D, \mathcal{P}, A, \mathcal{P}_1)$. % argument passing
- (4) $bc(D_1; D_2, \mathcal{P}, A, \mathcal{P}_1)$ if $bc(D_1, \mathcal{P}, A, \mathcal{P}_1)$. %
- (5) $bc(D_1; D_2, \mathcal{P}, A, \mathcal{P}_1)$ if $bc(D_2, \mathcal{P}, A, \mathcal{P}_1)$. %
- (6) $bc(uchoo(D_1, \dots, D_n), \mathcal{P}, A, \mathcal{P}_1)$ if choose a successful one $bc(D_i, \mathcal{P}, A, \mathcal{P}_1)$
- (7) $ex(\mathcal{P}, A, \mathcal{P}_1)$ if $D \in \mathcal{P}$ parand $bc(D, \mathcal{P}, A, \mathcal{P}_1)$. % a procedure call or a boolean condition.

- (8) $ex(\mathcal{P}, true, \mathcal{P})$. % True is always a success.
- (9) $ex(\mathcal{P}, x = E, \mathcal{P} \uplus \{\langle x, E' \rangle\})$ if $eval(\mathcal{P}, E, E')$. % the assignment statement. Here, \uplus denotes a set union but $\langle x, V \rangle$ in \mathcal{P} will be replaced by $\langle x, E' \rangle$.
- (10) $ex(\mathcal{P}, G_1; G_2, \mathcal{P}_2)$ if $ex(\mathcal{P}, G_1, \mathcal{P}_1)$ seqand $ex(\mathcal{P}_1, G_2, \mathcal{P}_2)$. % sequential composition

If $ex(\mathcal{P}, G, \mathcal{P}_1)$ has no derivation, then the machine returns the failure.

3. Examples

As an example, consider a simple smartphone which performs only two mutually exclusive tasks. The types of smartphone tasks are 1) play music with the speaker on, and, 2) sleep with the speaker off. An example of this object is provided by the following code where the program \mathcal{P} is of the form:

```
module smartphone
uchoo(speaker == on, speaker == off);
playmusic(x) = speaker == on; play music x hours;
sleep(y) = speaker == off; sleep y hours
```

and the goal G is of the form:

```
while true
playmusic(10);
sleep(14);
endwhile;
```

In the above, the machine plays the music for ten hours by turning on the speaker. After ten hours of playing, the machine sleeps for fourteen hours by turning off the speaker. Then the execution will repeat it again.

4. Conclusion

In this paper, we extend the core C with the addition of conjunctive statements within a class definition. This extension allows statements of the form $uchoo(D_1, \dots, D_n)$ where each D_i is a definition statement. This statement makes it possible for the core C to model decision steps from the machine.

5. Acknowledgements

This work was supported by Dong-A University Research Fund.

References

- [1] G. Japaridze, “Introduction to computability logic”, Annals of Pure and Applied Logic, vol.123, pp.1–99, 2003.
- [2] G. Japaridze, “Sequential operators in computability logic”, Information and Computation, vol.206, No.12, pp.1443–1475, 2008.

- [3] Lynn Andrea Stein, “Interactive programming: revolutionizing introductory computer science,” *ACM Comput. Surv.* 28, 4es, Article No. 103, December 1996.
- [4] Roly Perera, “First-order interactive programming,” 12th international conference on Practical Aspects of Declarative Languages (PADL’10), Springer-Verlag, Berlin, Heidelberg, pp. 186-200, Jan. 2010.
- [5] Avinash C. Kak, *Programming with Objects: A Comparative Presentation of Object-Oriented Programming with C++ and Java*, John Wiley & Sons, Inc., New York, NY, USA, 2003.
- [6] Joseph Albahari, and Ben Albahari, *C# 5.0 Pocket Reference*, O’Reilly Media, 224 pages, May 2012.
- [7] Joshua Bloch, *Effective Java*, Second Edition, Addison-Wesley, 346 pages, May 2008.